# 日本語言語資源の相互運用

狩野　芳伸[†]　　橋田　浩一[‡]
[†]東京大学 情報学環　kano@is.s.u-tokyo.ac.jp
[‡]産業技術総合研究所 社会知能技術研究ラボ　hasida.k@aist.go.jp

## 1　はじめに

近年、一般に利用可能な日本語の言語資源(注釈つきコーパスやソフトウェアツール)が充実しつつある。基盤的な言語資源の場合、再利用や組み合わせが頻繁に必要となり、相互運用性が求められる。しかしながら、言語資源は異なる組織・異なる時期において開発され、互換性がないことが多い。そこで、科研費特定領域研究「日本語コーパス」の成果を中心に、日本語言語資源間の相互運用をUIMA/U-Compareに準拠して実現した。データの処理やソフトウェアツールの実行の相互連携を極力自動化して容易にしている。これら互換化実装そのものに加え、実装の方法を調達仕様書のテンプレートとして公開し、多くの利用者による多様な資源の互換化作業を省力化することにより、仕様の明確な資源の相互運用が自然に普及するように配慮した。

## 2　背景

### 2.1 UIMA

UIMA (Unstructured Information Management Architecture)[1]とは、非構造化データのための相互運用性を提供するフレームワークである。UIMA は国際標準化団体 OASIS により承認された国際標準であり、その実装は Apache UIMA[2]としてオープンソースで公開されている。学術・産業の双方で利用者は増加しており、特に自然言語処理で広く利用され、UIMA 互換言語資源は数多く公開されている。

UIMA はXML 記述によるプログラミング言語非依存のメタデータ定義をさまざまなレベルで提供しており、API は Java および C++のものが存在する。その設計はコンポーネント志向というべきもので、あらゆる処理は UIMA コンポーネントからなる workflow の実行により行われる。コンポーネントは入れ子にでき、子の実行順序はプログラマブルで動的な制御も可能であるため、理論的にはほとんどあらゆる順序が実現できる。コンポーネントはウェブサービスとしても展開でき、ローカルサービスと混在させて透過的に実行できる。大規模データに対応した並列処理も可能である。

UIMA コンポーネントはCAS と呼ばれる構造体を受け取り、必要に応じ処理結果等を追加して返す。CAS は生テキストと付加データ(以下アノテーションと呼ぶ)から成り、アノテーションは相互に参照可能なため任意のグラフ構造を表現できる。アノテーションは明示的に型付けされる。Type system と呼ばれる型階層の定義は XML で記述され、開発者が提供する必要がある。アノテーションは生テキスト中の文字位置を用いてテキストと関連づけすることを想定している。これは stand-off annotation style と呼ばれ、XML のような in-line style よりも重層的なデータの取り扱いが容易である。

### 2.2 U-Compare

UIMA は優れたフレームワークであるが、あくまでフレームワークであるため、コンポーネントそのものや型定義は標準としては提供されない。また、基本的に開発者を想定しており、一般ユーザにとって UIMA を用いたシステムを構築するのはそれほど容易ではない。

我々はそういった UIMA の不足点を補い必要な機能を実装した統合自然言語処理システム U-Compare[3]を開発しオープンソースライセンス(LGPL)で公開している[4]。U-Compare の狙いの一つは、ユーザや開発者の負担を軽くするために、ツールへのアクセス・組み合わせ・実行を相互運用性に基づき徹底して自動化することとである。もう一つの狙いは、単に既存のものをつなげる以上に、比較・評価・解析・視覚化といった自然言語処理で必要とされる豊富な機能を UIMA 準拠で提供することにある。U-Compare はおおまかには platform 部分と互換 component 群に分かれる。

互換 component 群はすべて U-Compare type system に互換であり、入出力条件さえ満たせば単に実行順序を指定するだけで実行できることを保証している。Component 群は以下に述べる platform とは独立に使用可能である。U-Compare ではこれまで提供していたのは英語の言語資源のみであった。

platform 部分は、任意の UIMA コンポーネントに対し、workflow 作成 GUI、比較・評価機能、結果の視覚化など自然言語処理に必要な様々な機能を提

供している。これらの機能は汎用であり、以下に述べる日本語資源であってもそのまま利用できる。

platform を通じて U-Compare の component を利用する場合、platform のみならず component についても、インストール・更新・実行が自動的に行われる。既存の component を組み合わせて使うだけであれば、プログラミングは全く不要である。一方でコマンドラインモードも用意されており、GUI で作成した workflow をコマンドラインツールとして実行するといったことも可能である。

既存ツールを UIMA 対応にする場合、多くは入出力形式の変換が必要で、これを wrapping と呼ぶ。通常は UIMA の Java または C++ API を用いる。

他の言語で実装されたツールの場合や、ソースコードの改変にコストがかかる場合のために、我々は標準入出力経由でオリジナルのツールとやりとりし UIMA コンポーネント化する native tool wrapper を開発した。この wrapper はツールを実行するプロセスのリカバリ機能も備えている。これを用いれば開発者はフォーマット変換部分のみを実装すればよい。標準入出力経由であれば言語非依存であるため、Java でのコーディングをしなくともよいように、スクリプト言語で扱いやすい標準形式も用意した。これにより変換処理を好みの言語で行うことができる。このように標準入出力を用いると、実行時にプロセスが分離されエラーが波及しないことと、入出力形式さえ同じならば指定するコマンド名を変えるだけで wrap するツールを変更できるという利点もある。

## 3 日本語言語資源の相互運用

既存の言語資源の相互運用を考える場合は、以下のような点を考慮する必要がある。

まず、相互運用という場合、たいていはなんらかの変換処理が必要になる。そのときに、表現形式が変わってもオリジナルの情報を失うことがないよう、復元するのに十分な情報を保持すべきである。

次に、stand-off style を用いるからには、もともとのテキストをそのまま保持し、ツールの処理結果は極力アノテーションの側で扱うのが望ましい。ほとんどのツールは本質的に元テキストに改変を加えることはないからである。テキスト部分を不変とすれば、各ツールの影響の範囲が明確になると同時に、本来不要な前・後処理も考える必要がなくなり、入出力条件が簡素化される。たとえば、改行や空白はツールによっては無視されたり特殊な扱いを受けたりするが、こういった特殊文字もオリジナルのまま保存するようにする。

また、ユーザ・開発者からみたデータ構造のわかりやすさも考慮する必要があると考える。たとえば形態素列を取得したい場合、形態素型のオブジェクトを順に反復処理するのがわかりやすいが、コーパスによっては形態素的なものが平行して二種類ついていることもある。適切に型と型階層を定義することで、プログラムからの読み込みを容易にすることが必要であろう。

UIMA におけるデータ処理の単位は常に CAS であるが、CAS がテキストのどの部分を保持するべきかは特に定められていない。分散処理の可能性など処理効率を考えれば処理単位は小さいほどよいが、UIMA 枠内で複数の CAS にまたがる依存関係を扱うのは困難であるため、依存関係が閉じた最小のテキスト領域を CAS に対応させるのが妥当である。

UIMA コンポーネントを作成する際は、その機能単位に注意する必要がある。UIMA はあくまで枠組みでありどのような実装も可能であるが、コンポーネントが再利用を前提にしたブロックであると考えると、既存ツールをそのままコンポーネントにするのは適切でないことも多い。再利用という観点からすると、より小さな機能単位に分割したほうが理論的には再利用の可能性が高まる。しかし、小さすぎると入出力条件が複雑になりがちで、組み合わせての再利用がかえって難しくなる。我々は、入出力条件が入出力 type の静的リストで表現できる機能単位であるという条件下で、極力小さな機能単位に分割し、コンポーネント化している。

type と type system の定義はこれらの点を考慮しつつ、対象とする言語資源で必要な概念を十分表せるよう定義しておく必要がある。

こういった点を踏まえて、我々は日本語言語資源のうち再利用される可能性の高い基盤的な注釈付きコーパスやツール群を UIMA コンポーネント化した。その実装は U-Compare ウェブサイトでオープンソースで公開されている[5]。U-Compare platform を通じた利用であれば、他のコンポーネントの利用と同じくマウス操作だけで完結する。

### 3.1 Type Systemの設計

全体に共通して用いられる type として、文境界を表す `Sentence`、形態素を表す `Morpheme`、係り受けを表す `Dependency`、述語項構造を表す `FrameRelation`、モダリティ情報を表す `Modality` を定義した。それぞれオリジナルの情報を復元できるよう必要なフィールドセットが定義されている。特定の言語資源で用いられる type については以下各言語資源の項で触れる。

また、id 値による XML タグ間参照や、in-line スタイルで間接的に表現されていたアノテーション間の関係を、直接的にリンクとして表現している。たとえば UIMA Java API を使用して `Dependency` から係り先 `Morpheme` を取得するには、`getTarget` メソッドを呼び出せばよい。

### 3.2 日本語コーパス (BCCWJ)

代表性を有する大規模日本語書き言葉コーパス(以下では「日本語コーパス」という)[6]は、国立国語研究所を中心に開発されている 1 億語を超える規模の均衡コーパスである。そのうち一部のテキストについては、形態素・係り受け・述語項構造・モダリティ等の情報が付加されている。形態素および拡張モダリティ情報については XML 形式で表現されており、これを読み込める BCCWJReader を開発した。係り受けについては後述の Cabocha と同形式であり、述語項構造については後述の NAIST テキストコーパスと同形式であるため、同じ実装が使用できる。

### 3.3 京都大学テキストコーパス (Ver.4)

京都大学テキストコーパスは、毎日新聞記事計約 4 万文に対して形態素・構文情報を付与したもので、うち 5,000 文に対しては、格関係、照応・省略関係、共参照の情報が付与されている。前者は一般的な非交差係り受け情報であるが、後者はそれと平行して関係情報が追加されており、形態素境界も必ずしも一致しない。我々はこれらすべての情報を読み込める KyotoCorpusReader を開発し、それぞれを別個の系列として格納するようにした。

### 3.4 NAISTテキストコーパス (1.5)

NAIST(奈良先端科学技術大学院大学)テキストコーパス[7]は、毎日新聞記事計約 4 万文に対して格・共参照関係・照応関係の情報を付与したコーパスである。情報としては京都大学コーパスに類似のものといえるが、フォーマットが異なるため、別途 NaistCorpusReader を作成した。

### 3.5 GDAコーパス

GDA (Global Document Annotation, 大域文書修飾)[8]とは、統語的依存関係、代名詞等の照応、共参照、多義語の語義など、広汎な言語情報を XML で表現可能なフォーマットである。代表的な GDA 形式のコーパスとしては、毎日新聞 3000 記事に対するアノテーションが GSK より公開[9]されている。

GDA のアノテーションは他に比べ非常に細密であり、タグの種類も膨大である。形態素は他と同等であるため `Morpheme` を用いたが、他の情報については GDA 向けに別途 type をいくつか定義し、大まかなタグ種以外は文字列フィールドとしてタグ種名を保持した。

GDA の特徴の一つは、人間のアノテーション作業をサポートするために、(復元可能な)省略を許すなど工夫がされている点にある。UIMA 読み込み後の使用は機械的なものが主であるため、こうした省略は復元して格納した。また、GDA における統語的な構造は句構造であり、深い入れ子になっているが、その親子関係は in-line style で間接的に表現されている。これを UIMA 側に読み込む際は明示的な親子関係を抽出し表現した。親子関係は U-Compare の木構造表示コンポーネントと組み合わせれば視覚化することもできる。

### 3.6 形態素解析ツールChasen

Chasen[10]は NAIST 松本研究室で開発された、品詞付けを含む形態素解析ツールである。我々は native tool wrapper を用いて ChasenWrapper コンポーネントを作成した。Chasen は品詞以外にも基本形や読みの判定を行うため、出力される情報はすべて `Morpheme` のフィールドに格納した。

Chasen はルールベースで文境界を判定し、その結果を(デフォルトでは)改行として出力する。stand-off style という観点からは、元テキストに修正を加えずあくまで stand-off ポジションで表現したいので、文境界は `Sentence` により明示的に出力し、後段の処理で文境界を分ける必要があるときはこれを用いて wrapper 内で適宜処理することとした。

### 3.7 係り受け解析ツールCabocha

Cabocha[11]は Support Vector Machine を用いた係り受け解析器である。我々は、Cabocha の入出力指定オプションを固定した上で、native tool wrapper を用いて、`Morpheme` をうけとり `Dependency` を返す UIMA コンポーネント CabochaWrapper を作成した。上述のように内部的に `Sentence` を用いて入力を区切ったうえで処理している。

### 3.8 アノテーションツールChaki

Chaki (茶器)[12]は NAIST 松本研究室で開発されているアノテーションツールで、特に形態素や係り受け関係の編集が容易に行えるよう設計されている。Chaki では拡張 Cabocha 形式ファイルでのインポートおよびエクスポートが可能であり、この形式で読み書きをする UIMA コンポーネント ChakiReader および ChakiWriter を開発した。拡張 Cabocha 形式

では、Cabocha の扱う形態素と係り受け情報に加え、`Group`, `Link`, `Segment` によるタグを用いた拡張がなされており、日本語コーパスに含まれている交差した係り受けを表現できる。これら三種の情報を表せるよう type system を拡張した。

今後、Chaki は東京工業大学徳永研究室で開発されているアノテーションツール Slate とも互換化される予定である。

### 3.9 中納言

中納言[13]は国立国語研究所で開発されたコーパス検索システムで、Web アプリケーションとして公開されている[14]。中納言での検索は形態素解析を前提としており、独自の入力形式がある。我々は UIMA 側の `Morpheme` 列を中納言で読み込み可能な形式のファイルに変換し保存する UIMA コンポーネントとして ChunagonWrapper を開発した。

## 4 実装と調達仕様書のテンプレート化

我々の目的は、即座に利用可能な互換コンポーネントを提供するのに加え、第三者が新たな互換コンポーネントを容易に作成できるようにすることにある。

前述の互換化された言語資源群は、日本語の自然言語処理においてもっともよく使われるであろう形式やデータタイプをカバーしている。また、英語を含め一般に言語処理でよく用いられる形式の多くについて、読み書きできるコンポーネントを U-Compare から配布している。これらの実装をテンプレートとして再利用すれば、大概の言語資源互換化作業はごく一部の修正で済む。

現在の type system で対応しきれていない type が必要なときは、新たに定義する必要がある。意味的な互換性を保つためには type の互換性が必要であるため、第三者が新たに定義する際は必要に応じて我々も型設計作業をサポートしたいと考えている。

さらに、互換化したい既存言語資源の形式が明確に定義されているのであれば、実装作業は比較的単純であり、簡単な調達仕様書を書けば発注も容易であるので、そのテンプレートを用意した。

## 5 おわりに

我々は主要な日本語言語資源について、国際標準 UIMA に準拠して互換コンポーネント群を作成公開した。これらの組み合わせと実行にプログラミング作業は不要である。これらは単独で利用可能であるが、任意の UIMA コンポーネントに対応した言語処理システム U-Compare にも統合し、U-Compare の提供する様々な機能と共に簡単に言語資源を使用できるようにした。また、ソースコードと調達仕様書をテンプレートとして公開し、新たな互換コンポーネント作成の際の省力化を図っている。今後はさらなる日本語言語資源の追加や、基盤機能の拡張を予定している。


## 謝辞

本研究の一部は、科学研究費補助金特定領域研究「日本語コーパス」および基盤研究 C(21500130)の助成を受けて行われた。作業において多大なご協力をいただいた「日本語コーパス」ツール班と関係者の皆様方、特に松本裕治氏(NAIST)、森田敏生氏(総和技研)、山崎誠氏(国語研究所)、中村壮範氏(国語研究所)、徳永健伸氏(東京工業大学)、Dain Kaplan 氏(東京工業大学)、乾健太郎氏(東北大学)、小町守氏(NAIST)、松吉俊氏(NAIST)には深謝申し上げたい。


## 参考文献


[1] D. Ferrucci, A. Lally, D. Gruhl, E. Epstein, M. Schor, J. W. Murdock, A. Frenkiel, E. W. Brown, T. Hampp, Y. Doganata, C. Welty, L. Amini, G. Kofman, L. Kozakov, Y. Mass, *Towards an Interoperability Standard for Text and Multi-Modal Analytics,* RC24122, IBM Research Report, 2006.

[2] Apache UIMA. http://uima.apache.org/

[3] Y. Kano, W. A. Baumgartner, L. McCrohon, S. Ananiadou, K. B. Cohen, L. Hunter, J. Tsujii, “U-Compare: share and compare text mining tools with UIMA,” *Bioinformatics,* vol. 25, no. 15, pp. 1997-1998, 2009.

[4] U-Compare: UIMA-based integrated NLP system. http://u-compare.org/

[5] U-Compare 日本語ページ. http://u-compare.org/japanese.html

[6] 前川喜久雄, “代表性を有する大規模日本語書き言葉コーパスの構築,” *人工知能学会誌,* vol. 24, no. 5, pp. 616-622, 2009.

[7] NAIST テキストコーパス. http://cl.naist.jp/nldata/corpus/

[8] GDA (大域文書修飾). http://www.i-content.org/gda/

[9] GSK 新聞記事 GDA コーパス 2004. http://www.gsk.or.jp/catalog/GSK2009-B/catalog.html

[10] 形態素解析器 Chasen. http://chasen-legacy.sourceforge.jp/

[11] 日本語係り受け解析器 Cabocha. http://chasen.org/~taku/software/cabocha/

[12] アノテーションツール茶器 (Chaki). http://sourceforge.jp/projects/chaki/

[13] 中村壮範, 小木曽智信, 小椋秀樹, 小磯花絵, "Web 版コーパス検索アプリケーション「中納言」の公開," *特定領域研究「日本語コーパス」平成 21 年度全体会議予稿集*, 2009, pp. 107-110.

[14] コーパス検索アプリケーション「中納言」. http://morph.kotonoha.gr.jp/chunagon/